Operating Systems
C. Weissman Editor

On a Solution to the Cigarette Smoker's Problem (without conditional statements)

D.L. Parnas
Technische Hochschule Darmstadt

This report discusses a problem first introduced by Patil, who has claimed that the cigarette smoker's problem cannot be solved using the P and V operations introduced by Dijkstra unless conditional statements are used. An examination of Patil's proof shows that he has established this claim only under strong restrictions on the use of P and V. These restrictions eliminate programming techniques used by Dijkstra and others since the first introduction of the semaphore concept. This paper contains a solution to the problem. It also discusses the need for the generalized operators suggested by Patil.

Key Words and Phrases: operating systems, cooperating processes, process synchronization primitives
CR Categories: 4.3

This work was supported in part by the Naval Research Laboratory under project WF-15-241-601 and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited. Author's address: Research Group on Operating Systems I, Fachbereich Informatik, Technische Hochschule Darmstadt, Steubenplatz 12, Darmstadt, West Germany.

[1] In the present author's opinion, X, Y and Z could be combined first with each other and then with s without essentially changing the problem. Then the three "beta" processes could be eliminated. We leave them unchanged because the problem definition has the agent unchangeable.

Introduction

In a widely circulated and referenced memorandum [1, 2], Suhas Patil has introduced a synchronization problem called the "cigarette smoker's problem" and claimed that the problem cannot be solved using the P and V primitives introduced by Dijkstra [3] unless conditional statements are also used. On the basis of an elaborate proof in terms of Petri nets, it is concluded that the P and V primitives are not sufficiently powerful and that more complex operations are needed. This paper presents a solution to the cigarette smoker's problem without conditional statements, explicitly states an implicit assumption used in Patil's proof, and discusses the need for more powerful operations.

While a full introduction into the problem can be found in Patil [1], the aspects of the problem essential to this paper are related below.

There are three resource classes and also three processes using those resources. At a certain point in each process it must have one resource of each class in order to proceed. If it does not have all three, it must wait before proceeding. However, each of these processes has a permanent supply of one of the three resources. (No two of them have a permanent supply of the same resource.) There is a set of processes known as the agent which occasionally makes two of the resources available. The one process which has the remaining resource can then proceed. When it has finished with the three resources it is to notify the agent. The agent will not supply any more resources until it receives such notification.

In Patil's description of the problem the agent is considered to be an unchangeable part of the problem definition and is defined by the six processes shown below (notation is changed to be consistent with other *Communications* papers).

```
r_a : P(s);      r_b : P(s);      r_c : P s ;
      V(b);            V(a);            V(a);
      V(c);            V(c);            V(b);
      go to r_a ;      go to r_b ;      go to r_c ;

β_x : P(X);      β_y : P(Y);      β_z : P(Z);
      V(s);            V(s);            V(s);
      go to β_x ;      go to β_y ;      go to β_z ;
```

initially $s = 1$ and a, b, c, X, Y and $Z = 0$.

The semaphores a, b, and c are used by the agent to report the arrival of the three resources. Each semaphore denotes one of the resources. s is a mutual exclusion semaphore to assure that only one pair of resources is deposited at a time. X, Y, Z are to be used by the resource users to report that they are done with the resources.[1]

The problem is stated so as to allow the definition of any number of additional semaphores and processes and to use any number of P and V statements to write programs for the resource users which will use the resources and be free of deadlock. Conditional statements are not allowed in these programs. Patil states that this problem has no solution.

The Solution (processes and semaphores are in addition to the agent)

```
semaphore mutex; (initially 1)
integer t; (initially 0)
semaphore array S[1 : 6]; (initially 0)

δa : P(a);          δb : P(b);          δc : P(c);
     P (mutex);           P (mutex);          P (mutex);
     t ← t + 1;           t ← t + 2;          t ← t + 4;
     V(S[t]);             V(S[t]);            V(S[t]);
     V (mutex);           V (mutex);          V (mutex);
     go to δa ;           go to δb ;          go to δc ;

αx : P(S[6]);       αy : P(S[5]);       αz : P(S[3]);
     t ← 0;               t ← 0;              t ← 0;
     . . .                . . .               . . .
     V(X);                V(Y);               V(Z);
     go to αx ;           go to αy ;          go to αz ;

Optional: if overflow is a problem

δ1 : P(S[1]);       δ2 : P(S[2]);       δ3 : P(S[4]);
     go to δ1 ;           go to δ2 ;          go to δ3 ;
```

The · · · stands for some operations performed by the process.

Comments

The last three processes are superfluous unless one worries about the ill-defined problem of semaphore overflow. The solution works by simulating a six-branch case statement using a semaphore array and simple arithmetic operations. The solution is simpler than the published solution [1] using conditionals. P.J. Courtois [6] has shown that the extra processes could be eliminated by decrementing those semaphores in other processes and adding a semaphore $s[0]$.

On Patil's Proof

Patil [1] gives a method of representing cooperating processes as Petri nets, stating that, "the 'transition' representing the instruction $P[S]$ has an additional arc from the place corresponding to semaphore S and from the transition corresponding to the instruction $V[S]$, there is an additional arc to the place corresponding to the semaphore S." Every element of a semaphore array must be represented by its own place. The transitions corresponding to V operations may now place markers in any one (*but only one*) of these places. The description of the action of a Petri net [1] states that a transition places markers on *all* of its output places.

It appears then that Patil has used (but not stated) an assumption that there are no semaphore arrays. Since these arrays appeared in the earliest literature on the subject [3], the limitation reported by Patil is not a limitation on the primitives as they were described by Dijkstra.

Patil's Result

Patil has obtained the following result: A solution to the cigarette smoker's problem cannot be composed given:
(a) P and V operations on single semaphores—each operation appearing in the text operating on a specific semaphore determined by the text and fixed throughout execution of the program.
(b) Algol-like sequencing rule with no conditional statements.
(c) Assignment statements with arithmetic expressions (perhaps allowing quite restricted function calls), but no conditional expressions.
(d) A set of parallel processes (any number) each written from elements (a)–(c) above.

This result, although more restricted than one would like, is a definite contribution to the state of our knowledge about P and V.

On a Complication Arising from the Introduction of Semaphore Arrays

The use of semaphore arrays does introduce one minor complication which has not been discussed in the literature. In a call of the form "$P(S[E])$" (where E represents an arbitrary integer valued expression), the evaluation of E must take place only once and *before* execution of the body of "P". The evaluation is not considered a part of the "primitive" P. (In other words, we must be able to consider the state of the system during evaluation of the expression, whereas we have no information about the state of the system during the execution of a "Primitive" such as "P" or "V". The semaphore value will be referenced at least twice. Those interested in programming languages might be intrigued by the fact that the classical parameter passing modes found in Algol 60 are inadequate for this purpose. Were we to want to write an appropriate "P" algorithm in Algol 60, it would require use of the format $P(s,E)$ where s is a semaphore array. Only in this way could we call E by value while being able to refer to s by name We consider this minor difficulty to be a quirk in the design of Algol 60 rather than any limitation of the concept of the semaphore.

On the Yet Unsolved Problem

It is not the purpose of this paper to suggest that there are no limitations to the capabilities of Dijkstra's semaphore primitives. There are limitations of the semaphore operations. While we disagree with the result claimed by Patil, we applaud his goal of a precise and substantial evaluation of the Dijkstra primitives.

It is important, however, that such an investigation not investigate the power of these primitives under artificial restrictions. By artificial we mean restrictions which cannot be justified by practical considerations. In this author's opinion, restrictions prohibiting either conditionals or semaphore arrays are artificial. On the other hand, prohibition of "busy waiting" is quite realistic.

The justifications given by Patil for eliminating conditional statements are not valid. Some charac-

teristics of the definition of the agent are also artificial. In fact, the agent could be modified to make the problem more difficult by removing the restriction that the agent throws down only two resources and waits until they have been taken before adding more [6]. Although we do not have a solution to that more difficult problem (without conditional statements), we do not conclude that the problem is unsolvable.

An investigation of these questions is difficult because (1) we have no firm definition of the set of problems which we wish to be able to solve, and (2) in using the *PV* system we have great freedom in the way that we assign tasks to processes and the way that we assign interpretations to the semaphores in use. Often problems which appear unsolvable are easily solved when additional processes or semaphores are introduced. It would be artificial to rule out such solutions in any investigation of the capabilities of semaphore primitives.

"*P*" and "*V*" are deliberately designed so that when there are several processes waiting for a "*V*" operation on a given semaphore, the choice of the process to be released by a "*V*" is not specified. This "nondeterminism" is advantageous in writing programs where schedulers are subject to change or programs are likely to be moved to another machine. On the other hand, it makes the solution of "priority problems" such as [4] more difficult. Perhaps exact solutions to some are impossible. The question remains open. Some recent work by Belpaire and Wilmotte [8] and Lipton [9] has helped narrow the issues.

On More Powerful Primitives

Patil's paper ends with the introduction of a "more powerful" primitive than the "*P*" and "*V*" primitives. His *P* can simultaneously decrement several semaphores when *all* are passable! He suggests that the necessity of such a primitive is supported by the inability of "*P*" and "*V*" to solve the smoker's problem. Such proposals are not new—a similar proposal was made when *P* and *V* operations were believed insufficient for problems similar to those solved in [4].

Such arguments are invalid since the problems can be solved. Further, we note that the generalized operation can be programmed in a straightforward way using *P* and *V* (and conditionals).

In this author's opinion, the use of the word "primitive" to describe routines which might have been called "monitor routines" or "operating system service calls" suggests that the programs implementing the operations be small and quickly executed. There are obvious practical advantages to placing such restrictions on a code which is the uninterruptable "heart" of an operating system. "*P*" and "*V*" have the desired properties, the generalized operations do not.

There is no doubt that the generalized operations, can be useful in describing certain processes. If one wants to describe such processes, one should build the operations, but they need not be built as "primitives." This attitude has been taken by the authors of [5] where some interesting new upper level operations are suggested.

Patil's generalized operations have been called "parallel" operations because they simulate simultaneous activities on many semaphore variables. They make it easy to describe a single process which does tasks which could have been done by several cooperating sequential processes. Often one finds possibilities for parallel execution within such processes; these possibilities cannot be exploited by a multiprocessor system because one does not assign two processors to the same process simultaneously.[2] Often, because of the potential parallelism within the process, the program describing the process becomes very complex. In this sense, the generalized operation is akin to the "go to" statement in programming languages. Both add nothing to the set of soluble problems; both make it easier to write programs which should not be written. Those who feel dissatisfied with the *P,V* primitive system could look at some " more primitive primitives" suggested by Belpair and Wilmotte [8] and Wodon (7).

Those interested in further discussions of the cigarette smoker's problem itself (including a correctness proof for the solution given here) should read [10].

Acknowledgment. I am indebted to Wing Hing Huen for helping me to strengthen one of the arguments in this paper. I am also grateful to P. Wodon and P.J. Courtois for helpful comments. An anonymous referee is acknowledged for his lucid description of the proper conclusion of Patil's paper.

Received 1972; revised July 1974

[2] Such a restriction is the operational definition of "process" or "task" in most operating systems.

References

1. Patil, S.S. Limitations and capabilities of Dijkstra's semaphore Primitives for coordination among processes. Proj. MAC, Computational Structures Group Memo 57, Feb. 1971.
2. Project MAC. Progress Report, 1970–71.
3. Dijkstra, E.W. Co-operating sequential processes. In *Programming Languages*, F. Genuys (Ed.), Academic Press, New York, 1968. [First published by T.H. Eindhoven, Eindhoven, The Netherlands, 1965.]
4. Courtois, P.J., Heymans, F., and Parnas, D.L. Concurrent control with readers and writers. *Comm. ACM 15* (Oct. 1971), 667–668.
5. Vantilborgh, H., van Lamsweerde, A. On an extension of Dijkstra's semaphore primitives. Rep. R192, MBLE, Laboratoire de Recherches, Brussels. Also published in *Information Processing Letters*, North Holland Pub. Co., Amsterdam.
6. Courtois, P.J. Private communication.
7. Wodon, P. Still another tool for synchronizing cooperating processes. Dep. Computer Sci. Rep., Carnegie-Mellon U., Pittsburgh, Pa., 1972.
8. Belpaire, G., and Willmotte, J.P. Proc. 1973 European ACM Symposium, Davos, Switzerland.
9. Lipton, Richard. On synchronization primitives. Ph.D. Th., Carnegie Mellon U., Pittsburgh, Pa., June 1973.
10. Habermann, A.N. On a solution and a generalization of the cigarette smoker's problem. Techn. Rep., Carnegie-Mellon U., August 1972.